RL-TR-95-3
Final Technical Report
January 1995

ADAPTIVE FAULT RESISTANT SYSTEM (AFRS)

Martin Marietta Laboratories - Moorestown

Bob R. Gupta, Angela M. Kitchen, and Don A. Sutton

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-95-3 has been reviewed and is approved for publication.

APPROVED: Jerry L. Dussault

JERRY L. DUSSAULT
Project Engineer

FOR THE COMMANDER: Henry J. Bush

HENRY J. BUSH
Acting Deputy Director
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (C3AB) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 1995	Final Jun 92 - Apr 94

4. TITLE AND SUBTITLE

ADAPTIVE FAULT RESISTANT SYSTEM (AFRS)

5. FUNDING NUMBERS

C - F30602-92-C-0097
PE - 63728F
PR - 2530
TA - 01
WU - 54

6. AUTHOR(S)

Bob R. Gupta, Angela M. Kitchen, and Don A. Sutton

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Martin Marietta Laboratories - Moorestown
Building 145
Moorestown Corporate Center
Moorestown NJ 08057

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Rome Laboratory (C3AB)
525 Brooks Rd
Griffiss AFB NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

RL-TR-95-3

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: Jerry L. Dussault/C3AB/(315) 330-2067

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This report summarizes the architecture and design developed for an adaptive fault-resistant distributed computing environment. A prototype Adaptive Fault Resistant System (AFRS) was developed as part of an ongoing investigation into using adaptive fault-tolerant techniques to improve the survivability of Battle Management/C3 systems. An Adaptive Fault Manager (AFM) is described. The AFM provides a capability to monitor and dynamically reconfigure a system in response to changes in its internal state (e.g., equipment failures) and/or changes that occur in the external environment. The enabling technologies that were exploited to implement this system are described, and a brief discussion of lessons learned is presented. A notional Battle Management/C3 application was selected and modified to demonstrate the adaptive fault-tolerance concept. That application, Theater Missile Defense - Command and Control (TMD-CC), is also briefly described.

14. SUBJECT TERMS

Distributed computing, Fault-tolerance, Adaptive fault tolerance

15. NUMBER OF PAGES

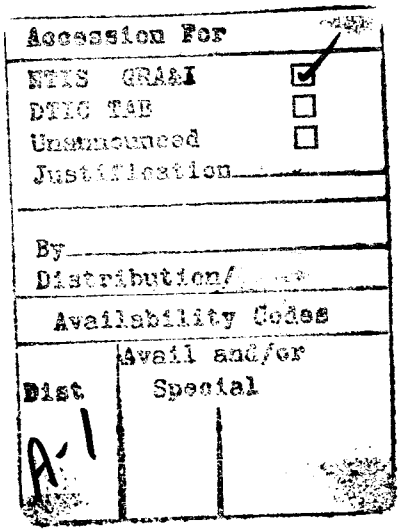
56

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures



1. Introduction

This report documents the work performed on the Adaptive Fault Resistant System (AFRS) contract (contract number F30602-92-C-0097), sponsored by the US Air Force's Rome Laboratory, Directorate of Command, Control and Communications, Computer Systems Technology Branch (C3AB) . The work documented in this report was performed during the period of July, 1992 until April, 1994.

1.1 Program Rationale

The objective of the AFRS program is to provide large complex distributed military systems with greater degrees of survivability, availability, and graceful degradation than is currently available.

Many systems currently meet high availability and reliability demands by using specific fault management strategies to detect and recover from potential problem areas. Most research on these systems has focused on the management of static threat and envir onmental conditions. Battle Management/Command, Control, and Communication (BM/C3) systems are designed to manage static threat and environmental conditions as well, even though they exist not in a static, but in a highly dynamic environment. The dynamics occur along several dimensions: external situation, fault and threat characteristics, fault and threat prediction, and resource limitations. Since continued effectiveness of BM/C3 systems is essential to our national security, survivability, non-catastro phic behavior, and the most effective use of limited resources are critical attributes that must be provided.

A static approach to fault management is inappropriate for BM/C3 systems because 1) requirements specified in the system's objective function and demands on system assets may change in response to changes in the dynamic operating environment; and 2) designing for worst case situation in every dimension of conceivable threat is cost prohibitive. An adaptive approach to fault management enables a system to dynamically tailor its fault management mechanisms to changes within the environment that influence its objective function, and to dynamically apply its limited assets appropriately.

The Adaptive Fault Resistant System must contain the ability to change a system's fault management mechanisms and modify fault management parameters at run-time to respond to changes in the dynamic environment and resource availability while maintaining the system's overall performance, consistency, and functional requirements. The adaptive fault resistant system must operate in a manner that coordinates all system resources toward a common fault management goal.

1.2 Report Organization

This report is organized into seven sections: (1) Introduction, (2) Overview of the Adaptive Fault Manager, (3) Theater Missile Defense Application, (4) Testbed Description, (5) Lessons Learned, and (6) Appendix.

Section 1 – Introduction includes topics on program rationale, report and program organization, team members, and significant events.

Section 2 – Overview of the AFM gives the technical detail of how the AFM was designed and built. It includes a background of the technologies and the need for an AFM as well as the design of all system components.

Section 3 – Theater Missile Defense Application contains the evaluation and selection criteria for the AFRS demonstration application. It also includes reasons why TMD was selected and a description of the demonstration.

Section 4 – Testbed Description provides the design, setup, and hardware/software requirements of the testbed which supports AFRS.

Section 5 – Lessons Learned covers issues which were discovered during the design and implementation of AFRS and includes technical observations by the engineers of AFRS as well as benefits and limitations of the AFRS technology.

Section 6 – Appendix includes any supporting material which would help the reader in understanding the work that was performed on the AFRS.

1.3 Program Organization

The AFRS program was built upon research into adaptive fault tolerance techniques performed under the Adaptive Fault Tolerance contract. The products of AFT research included, adaptive fault tolerance concept definition, system architecture specification, adaptive behavior management, fault taxonomy development, fault tolerance technique classification, investigation of potential adaptations, and the development of notional examples.

AFRS incorporated the results of AFT research into the AFRS component models and their implementation, further extended and modified the results of AFT research through the definition and implementation of resource management, and specified an architectural framework for building general adaptive fault resistant systems.

1.4 Team Members

The AFRS program is sponsored by the US Air Force Rome Laboratory. The technical efforts of the MML AFRS team are directed by Jerry Dussault of the Computer Systems Technology Branch (C3AB) of Rome Laboratory's Directorate of Command, Control and Communications. Martin Marietta Laboratories ▪ Moorestown is the AFRS prime contractor and program lead.

Other team members include the University of Maryland, BBN Systems and Technologies, and Len T. Armstrong, from BCI. The University of Maryland is developing MARUTI, a hard real-time, distributed, fault tolerant operating system. Many of MARUTI's concepts, such as "monitors" and "horizons," complement the research demands of the AFRS program. BBN Systems and Technologies, the developers of the Cronus system development environment, are providing Cronus support. Len T. Armstrong has been placed under contract from BCI for general AFRS technology design and implementation support. Mr. Armstrong was a principal participant in the design and implementation of the AFT program concept and demonstration.

1.5 Significant Events

The Initial Program Review was held at Rome Laboratory on December 10, 1992. MML ▪ Moorestown presented the baseline AFRS architecture and the design and implementation issues identified in each of the major components.

The AFRS concept was presented by Stephen Bate at the Rome Laboratory C3AB annual Technology Exchange meeting held January 1993, as a first step in accomplishing the AFRS technology transfer.

An MML ECP white paper in August, 1992 and a subsequent technical/cost proposal was submitted in November, 1992. The award of this ECP allowed MML to build a realistic demonstration of the AFRS technology.

The AFRS design and implementation effort was presented by Bob Gupta at the Rome Laboratory C3AB Technology Exchange meeting in October, 1993. A theoretical basis for AFRS and implementation experience was well received by the audience.

MML's AFRS team met with SRI's team to exchange technology and discuss results and future directions of fault tolerance at SRI in Princeton, NJ during October, 1993.

Rome Laboratory's Jerry Dussault and Pat Hurley met with MML's AFRS team in Moorestown in November, 1993 to discuss the implementation effort and begin the user training process.

2. Overview of the Adaptive Fault Manager

2.1 Background

AFRS technology is implemented in an application through the AFM. The AFM is a natural extension of the Adaptive Behavior Manager developed in the AFT program.

The AFM provides a fault management environment that meets a system's internal and external state demands across all system resources. To achieve this, the AFM dynamically alters the system's fault management mechanisms in response to changing system requirements. Since the fault management mechanisms place additional demands on system resources, integrated resource management is a critical component of adaptive fault management.

To meet the fault management demands of the AFRS program, the AFM is a combination of database, state observation, decision making, and resource control components.

The AFM must continually observe and evaluate a system's internal and external states. The internal state contains information over which the system has direct influence and control. Examples of internal state information include 1) hardware (processing load, virtual and physical memory usage, and communications loading/packet transfer rate), 2) application (application state and configuration), and 3) fault history (fault type and fault rate of recently experienced faults).

The external state contains information over which a system has no direct control, but that directly influences the operation and requirements of the system. External state information includes mission and mode of operation. An example of the mode element of an external state in a typical BM/C3 system is High Threat Mode. The external state places dynamic requirements on the internal state. As the external state changes during the operation of a system, new requirements are placed on the internal state.

At the highest level, system requirements are specified by an objective function consisting of performance, consistency, and functionality. For example, a BM/C3 application, in Low Threat Mode demands higher consistency and functionality, and lower performance. Conversely, High Threat Mode demands higher performance and lower consistency and functionality. This is because Low Threat Mode allows some deadlines to be missed (lower performance) so additional processing can be performed (higher functionality) for avoiding "false alarms" (higher consistency). High Threat Mode probability requires maximum attention to real-time deadline requirements (higher performance)

and less attention to false alarm checks (lower functionality, lower consistency). Some fault management mechanisms are better suited to Low Threat Mode requirements (e.g., recovery blocks), while others are better suited to High Threat Mode requirements (e.g., distributed recovery blocks).

2.2 AFM Architecture

This paragraph describes the Adaptive Fault Manager's system architecture. Figure 2-1 shows the architecture of the AFM.

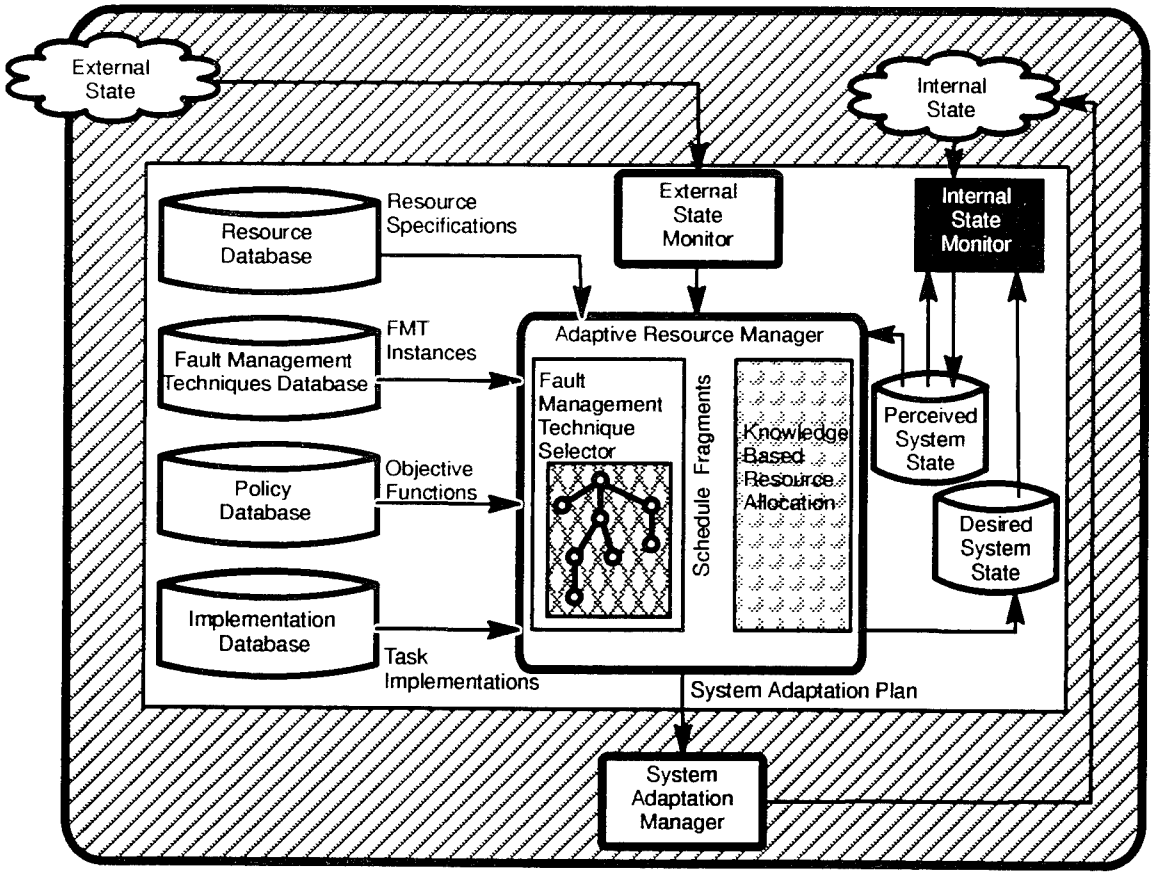


Figure 2-1 AFRS System Architecture Diagram

The internal state monitor is responsible for obtaining information about the system's internal state. The external state monitor is responsible for obtaining information about the system's external state. Although external state changes are outside the influence of the system, they are understood through application-defined methods. For example, user or sensor input may trigger (or verify) a change from Low Threat Mode to High Threat Mode.

To obtain cost/benefit information of all potential adaptations, the AFM uses four databases to model a distributed system's objective function and resource usage characteristics: a Fault Management Techniques database, a Policy database, an Implementation database, and a Resource database. The Policy database relates the external state of the system to the corresponding objective function requirements. For every mode and mission pair of the system, the policy database has corresponding required levels of functionality, performance, and consistency. The fault management techniques database provides a relationship between a system's available fault management mechanisms and their corresponding cost and benefits in the objective function (i.e., requirements) and resource usage domains. Since many fault management techniques are "wrapped around" specific application tasks, cost/benefit information on the tasks themselves is needed to perform efficient tradeoffs. The cost/benefit of a fault tolerant task is a function of the costs/benefits of both a specific fault management technique and specific implementation of the task that is used. The Implementation database provides cost/benefit information for the application tasks. Since each task may have multiple implementations in a given system (such as an implementation to run on a Connection Machine and an implementation to run on a Sun), there is not necessarily a one-to-one correspondence between tasks and their implementations. The resource database maintains information of the resources and their characteristic (i.e., memory, cpu, disk, etc.).

The Adaptive Resource Manager (ARM) is the remaining AFM component. The ARM determines when adaptations of fault management mechanisms are needed by evaluating internal and external state and performing objective function and resource management cost/benefit tradeoffs. The ARM also schedules fault tolerant tasks in the distributed environment.

The ARM accomplishes its complex goals through a heuristic cost/benefit evaluation of available task implementations and fault management mechanisms that can be used to achieve the system's objective function goals. Task implementations are paired with

possible fault management techniques and assigned to nodes in "Course- of-Action" trees. Costs (resource usage, performance) and benefits (consistency maintained, fault coverage percentage) are evaluated for all possible pairs (nodes) in the tree. Nodes that do not meet objective function requirements are discarded. The remaining nodes are assigned a "worth" value. The highest valued nodes are then sent to a knowledge-based resource scheduler.

The Adaptive Fault Manager (AFM) top-level data flow diagram is presented in Figure 2-2.

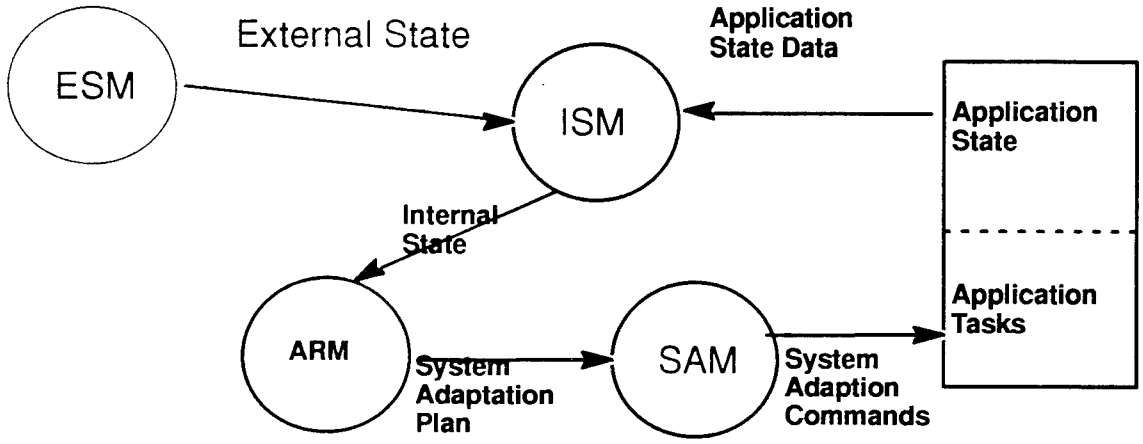


Figure 2-2 AFRS Data Flow

The first step in AFM processing is the generation of an adaptation plan. The adaptation plan generation is triggered by a change in the system's external state or by the reception of an alert condition generated by the detect anomalies component of the Internal State Monitor.

The Internal State Monitor generates alert conditions when the system's perceived internal state diverges from the desired internal state. The reception of an alert condition does not guarantee that an adaptation plan is produced by the ARM; it merely alerts the ARM that there may be a need for replanning the allocation of system resources.

If a change in the external system state triggers the retrieval of a new objective function set, then the new objective function is converted and stored into the Desired Internal State database.

If an adaptation plan is produced by the ARM, it is sent to the System Adaptation module. The arrival of an adaptation plan causes the Change Control module to construct and deliver adaptation commands to application tasks.

The application tasks are instrumented to provide state data which is continuously gathered and stored into the Perceived Internal State database. In addition, a set of default system and node level hardware and software information is gathered and stored into the Perceived Internal State database.

The ISM monitors the operation of the system against the desired system state. If the Perceived and Desired Internal State databases diverge past a threshold point, an alert to the ARM is generated and the cycle of adaptation and monitoring may begin again.

2.3 Design Philosophy

The design philosophy for the implementation of the AFM and for the representation of applications controlled and instrumented by the AFM can be described as a model-based, event-driven, feedback-controller system.

The AFM components and the application tasks themselves are modeled as system objects which provide specific services that are accessed through well-defined public interfaces. The modeled objects are described as event-driven because they passively wait for invocation of their services through input events. Invocation of the object's service produces outputs which may trigger invocation of a service on another object. Each AFRS system component has been described in terms of triggers, inputs, outputs, timing constraints and behavior.

Existing applications are encapsulated within a shell which is capable of discreetly providing information about the status of application specific and system generic data and processes. The monitored information is processed by the AFM, and may result in AFM directed adaptations in some or all of the system objects. These adaptations are invoked on the objects through the same shell which enables and supports the gathering of monitored data.

Monitored data may result in triggering a condition, such as an error rate threshold, that invokes an adaptation action. The AFRS does not only monitor state information, it also causes adjustments in task behavior. For this reason, the AFRS is viewed as a feedback-controller system. This design philosophy was selected because it is applicable to a broad range of existing applications and allows applications which were constructed under other paradigms to be encapsulated to fit into the AFM architecture. Use of this approach therefore provides a greater assurance that the resulting AFM is broadly applicable to a wide range of potential applications and is constructed using a generic, reusable architecture.

2.4 Design of AFM Components

2.4.1 Internal State Monitor (ISM)

The Internal State Monitor's (ISM) primary task is to build a picture of the system's state and to use this information to detect differences between the observed and expected states. Information gathering on a system wide level is difficult and must take into account the ordering of information as well as communication delays. By leveraging work done on the internal IR&D project Distributed Resource Manager (DRM), and MMLM's experience with ISIS/Meta[1], the ISM was built with efficient monitoring capabilities.

There are 3 methods which provide the ISM with the information it needs. The first is that the ISM can query the System Databases for information it needs to initialize itself. The second method is through sensors and actuators that are provided by the DRM to monitor and modify applications. And, the third method is by sending and receiving messages to/from the ARM. This method is the means by which applications and fault tolerant techniques are assigned to resources and are modified.

The Distributed Resource Manager (DRM), a state of the art tool for monitoring distributed systems, informs the ISM of the health of tasks, and the observed resource utilization of an application. DRM was developed using IR&D funds over several years and has been successfully used in many different projects and contracts. It is based on the ISIS and Meta toolkits from Cornell University and allows one to monitor many things of interest in a distributed system, including user state variables in applications and the

1. More information on ISIS/META can be found in sections 2.5.1 and 2.5.2.

allocation of resources such as CPU, communication, and memory. It also monitors failures of components in the system (networks, processors, storage). By using this established tool the ISM was able to leverage a good deal of its functionality.

Finally, the application itself may provide information to the ISM indicating its perceived ability to meet imposed requirements as well as other application specific attributes necessary to detect anomalies. The method by which the application informs the ISM is through META sensors which are incorporated into the application. The META sensors are available through the DRM as well as the AFRS Application Programmer's Interface thereby offering a consistent mechanism for communication.

These sources of information are then combined for all the tasks in the system by the ISM to form a picture of the distributed system's state. This state is used to detect anomalies.

When an anomaly has been detected it is the ISM's job to communicate the state of the world as well as the anomaly detected to the ARM which is responsible for correcting the situation.

2.4.2 External State Monitor (ESM)

MML used its experience in distributed system state monitoring gained in the DRM program to provide an efficient and easy-to-implement External State Monitor using the sensor concepts of the ISIS and Meta programming toolkits. The External State Monitor design is an extension of the Internal State Monitor design. It allows the sensed information to originate from programs outside of the AFRS system. This provides the capability for external programs to interface with the External State Monitor to provide external state information.

The AFM requires access to external state information to optimally match system resources and fault management technique selections to the requirements of the outside world. The External State Monitor must sense external information, such as mode of operation or mission state, and provide this information to the ARM.

The External State Monitor passes information between components of the system through a sensor namespace. These sensors reside in one or more programs that will either simulate the system's external state or allow the demonstrator to manually change it. In general, changes in the external state occur either through a manual hu-

man interface or under the control of application-dependent automated reasoning tasks. This design enables the External State Monitor to be notified automatically whenever any aspect of the external state changes so it can react to the new information.

An additional bonus of this design is that Meta allows functions within the External State Monitor to be called whenever any of the externally sensed values changes. The External State Monitor can be programmed to automatically provide information to the other parts of the AFM so they can react appropriately.

2.4.3 Adaptive Resource Manager (ARM)

The Adaptive Resource Manager (ARM) is responsible for responding to changes in the system state, evaluating the impact of the changes, and, if necessary, producing a system adaptation plan which mitigates the impact of the changes on the system. The ARM is activated by an alert from either the External or Internal State Monitors.

Examples of adjustments which may be recommended include: (1) adjusting the fault management technique, (2) altering resources consumption by deleting unnecessary tasks, or (3) selecting either a new task or fault management technique implementation. The Adaptive Resource Manager allows for adaptive modifications to the configuration of tasks and fault management techniques. The ARM retrieves information from the Policy, Fault Management Techniques, Implementation and Resource databases as well as from the Internal and External State Monitors to determine the current system requirements and the current state of system resources.

The ARM produces task execution and resource usage schedules which are delivered to the Change Control Module in real-time. The resulting schedules satisfy and maintain the performance, functionality and the consistency levels required as specified in the objective function set. Additionally, the ARM responds to ongoing fluctuations in the system's internal and external states and incorporates appropriate responses into the schedules. Internal state changes correspond to changes in the dynamic resource availability and external state changes correspond to new objective function requirements.

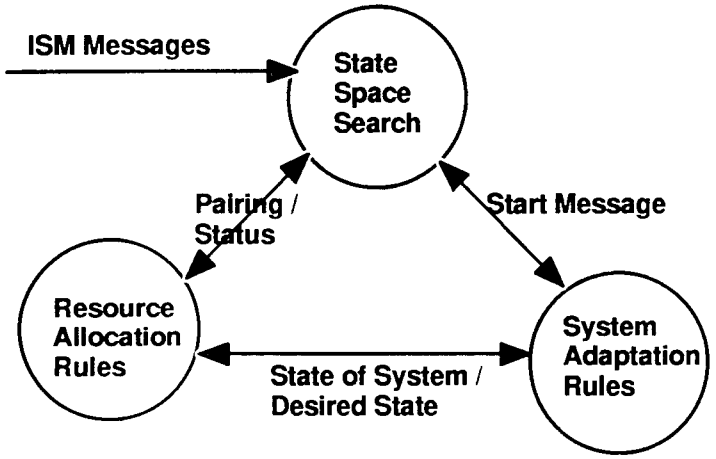


Figure 2-3 Adaptive Resource Manager Architecture

The Data Flow in the Adaptive Resource Manager has been refined as illustrated in figure 2-3. This new organization allows more flexibility in the design of the system. Through functional decomposition of the Adaptation task into 1) a Course of Action (COA) State Space Search module, 2) a Resource Allocation Rules and 3) a System Adaptation Planner rules, it is possible to develop the three functions independently. Well defined, consistent interface specifications describe inter-module communication. Independent development of each module allows for the enhancement or extension of any of the modules without affecting code already developed or requiring module re-design. For example, the Allocation Function could be extended to take into account reliability measures of system resources, or the COA State Space Search could be replaced with a more efficient search algorithm in the future.

2.4.4 System Adaptation Manager (SAM)

The System Adaption Manager's function in the AFRS is to receive the System Adaptation Plan from the ARM and actually make the system calls to implement that plan. It is the module responsible to make modifications to and start or stop processes. However, to accomplish this task, the SAM needs to know the same type and amount of information as the ISM. In other words, it needs to have all the same knowledge or state information and as much control as the ISM does in the system. For this reason, it was decided that the SAM would become incorporated into the ISM and become one of the functions of the ISM. Since the SAM still maintains a unique piece of functionality, the MML team feels that it warrants a high level description in the AFRS architecture. However, the design and implementation details have been included in the ISM section and

the reader is encouraged to reference that section for more detail.

2.4.5 Application Programmer's Interface (API)

Out initial design looked into trying to provide a non intrusive programmer's interface to the AFRS system by mirroring the kernel interface. This would have eliminated the need for programmers to learn about AFRS, but it quickly became so complicated with special cases and insufficient information that it had to be abandoned. A more traditional client library that offers a convenient interface for programmers to send asynchronous messages, and to request remote procedure calls (RPC) was then developed.

The Fault Management Application Programmers Interface (FMT API) had several design criteria: easy to use, functional interface independent of FMT being applied, expandable to new FMTs, and statefull verse stateless information. The state problem was addressed by shifting some requirements onto the user. By requiring the user to supply functions which encapsulate and interpret state information the user must decide which pieces of information need to be transferred between replicas because there is no way the system can. Guidelines and guarantees describing these requirements are provided. Common examples of statefull pieces of information the user must encapsulate are the current simulation time and the local database of information contained in global variables. Examples of information which shouldn't be transferred between replicas are file descriptors and performance information collected about the instance.

The heart of the FMT API evolved into 6 functions: AFRS_init (declares task name and the callback routines necessary for state transfer), AFRS_connect (declares intent to communicate with another task), AFRS_msg_declare (declares receivable messages types and their associated callback function), AFRS_begin (called at end of initialization), AFRS_send (sends an asynchronous message), AFRS_RPC (makes a remote procedure call to another task and returns its response), and AFRS_reply (sends response to an RPC call back to requestor). With only these 6 functions it is possible to produce a very powerful message passing system for use within the AFRS system. For a detailed description of these and other functions see the Software User's Manual Section 5.2.

The key advantages of our API include removing the constant re-implementation of

fundamental fault tolerant techniques designed to handle failures and anomalies while providing a message/RPC based paradigm upon which to build distributed applications. Currently 5 FMTs are supported: none, restart, primary/backup, N-modular-redundancy, and N-version.

2.4.6 System Databases

The function of the system databases is to provide a central repository of system data which can be accessed by all AFM modules requiring this information. The system databases have been designed using the Cronus toolkit to provide the distributed capability needed for AFRS. There are 4 distinct databases which are supported: (1) Policy, (2) Fault Management Techniques, (3) Implementation, and (4) Resource.

2.4.6.1 Policy Database

The Policy database lists tasks for each objective function by class name and priority. Each task represents a class of implementations which have corresponding performance, consistency, and fault management goals. The subset of tasks with the most critical priority represents the minimum acceptable configuration that must be available to accomplish the system's goals.

The Policy database defines performance, consistency, and fault management goals separately for each task. Performance goals define timing requirements including whether deadlines are hard or soft, periodicity and response time.

Figure 2-4 displays the layout of the policy database.

The consistency goals describe the degree of internal (application embedded) fault tolerance supported by the task and the degree of external (encapsulated around the application task) fault tolerance desired. Additionally, the consistency goals define acceptable error rates and types. The performance and consistency goals specified for each task in the Policy database are used in the Internal State Monitor's representation of the desired internal system state. Changes in the perceived internal state may then trigger alerts if the performance and consistency goals stated in the Policy database deviate across the defined thresholds.

Fault coverage goals describe whether software and/or hardware fault coverage is required, the number of simultaneous faults to be tolerated, and whether erroneous, omission, crash and/or deadline related faults should be covered.

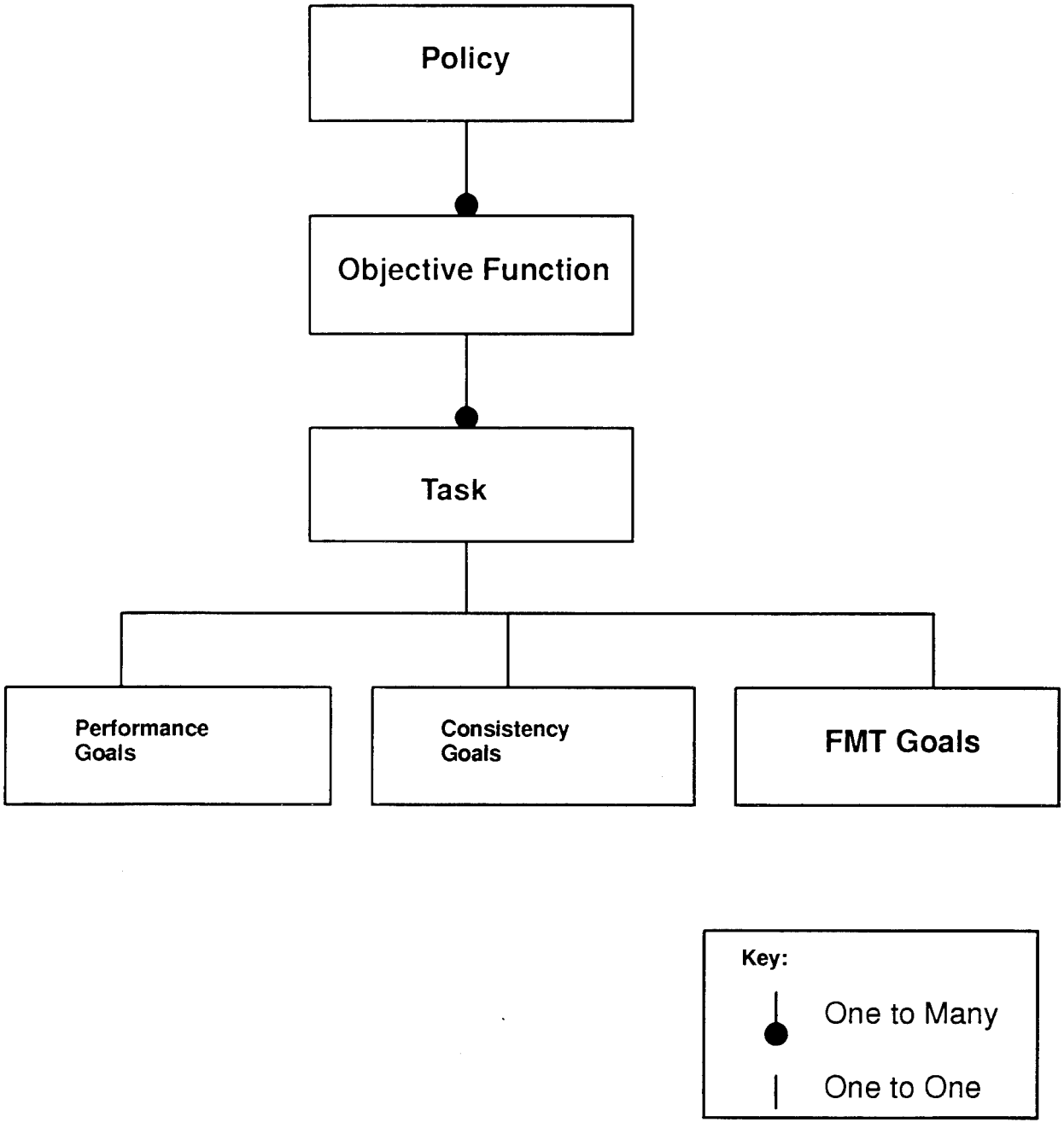


Figure 2–4 Policy Database Structure

Note that the fault coverage goal stated in the Policy database is a starting point or recommendation used in the selection among specific fault management techniques; additional information gathered at run-time may override the recommendation in the Policy database. For example, the Policy database may require fault tolerance only to hardware failures, excluding software related faults. Repeated erroneous results may require the transition to fault management techniques which guard against hardware and software faults. The Policy database is queried by the ARM when changes in the system's external state require the retrieval of a new objective function.

2.4.6.2 Fault Management Techniques Database

The Fault Management Techniques database provides a relationship between a system's available fault management mechanisms and their corresponding costs and benefits in both the objective function and resource usage domains. Each fault management mechanism in the database has a set of costs related to its overhead and benefits specifying consistency supported and fault coverage provided.

The ARM retrieves information from the Policy database including the expected fault mode for a given external state. This information is used in traversing a path through the Fault Management Techniques database to a set of fault management technique instances which are appropriate to the expected fault mode. For example, if multiple simultaneous software faults are expected in a given external state, the Fault Management Techniques database directs the ARM to a set of techniques which provide software fault coverage, such as recovery blocks or N-version programming. The fault mode model provides enough information to lead the ARM to a group of appropriate techniques, but it does not provide a mechanism to choose among the techniques or their implementations.

Selection of a specific fault management technique requires additional information including performance constraints, resource usage overhead costs, and recovery time. The costs of each fault management technique instance must be weighed against the benefits defined for that technique. Therefore, each technique also has benefits defined for the technique's degree of fault coverage and consistency maintenance.

The costs of each fault management technique instance must be weighed against the benefits defined for that technique. Therefore, each technique also has benefits defined for the technique's degree of fault coverage and consistency maintenance. The information needed to select a particular pairing of fault management technique instance with task implementation(s) is therefore derived from data provided in the Im-

plementation database, the Fault Management Techniques database, and on requirements obtained from the Policy database. The representation of the fault coverage, consistency, performance data is similar to the representation of those objects in the Policy and Implementation databases.

Additionally, fault management techniques are also grouped into classes within which possible adaptations are defined. Previous AFT studies showed that arbitrary adaptations between fault management techniques can not be efficiently performed. An understanding of similarities between techniques is needed to form logical groupings or classes. Similarities used to form classes include input structure, output structure, and desired fault tolerant capability.

An individual fault management technique may be a member of many classes. Each class contains a list of techniques that are members of that class. To generalize the fault management structure and provide a means of comparing a system with no fault management properties or static fault management against the AFRS system, a Null fault management technique and a Null fault management class are defined. The Null technique provides no additional overhead costs and no fault coverage or consistency maintained benefits. The Null technique allows fault management classes to contain techniques that allow the fault management to seamlessly degrade completely. The Null class is a special class that contains only the Null technique. The inclusion of the Null class allows all tasks within a system to specify a desired level of fault tolerance, even if that level is null.

2.4.6.3 Implementation Database

The Implementation database defines alternate implementations of application tasks and characterizes them in terms of resource costs and policy benefits. The ARM requires this database to determine the resource costs and policy benefits of application tasks. The Implementation database also provides the requisite information needed to execute and control application tasks.

The Implementation database is relevant to both external and internal state changes. For instance, a change in system mode may require that a particular task, such as tracking, be performed more frequently but less accurately. The database records whether there is an implementation that runs faster with less precision.

On the other hand, the internal state monitor may observe an unacceptable increase in the error rate for a task, such as data transmission over a satellite to ground station link.

The Implementation database would be consulted for an alternate communications protocol implementation, along with measures of its reduced error rate at the expense of a reduction in throughput.

Examples of different implementations of tasks in the same class include, multiple implementations tailored to specific processors or computational paradigms such as a shared memory multi-processor versus a uni-processor implementation, and task implementations with varying degrees of precision and different performance rates.

While costs and benefits of fault management techniques are listed in the Fault Management Techniques database, costs and benefits of each available implementation of each application task are located in the Implementation database. Many fault management techniques, such as N-modular redundancy, replicate tasks. Total processing, communications, and data storage costs, as well as performance and consistency benefits, are a function of the requirements of the task implementation and of the fault management technique. Together, the Implementation, Fault Management Techniques and Resource databases, provide all the information necessary to make resource tradeoffs in choosing combinations of task implementations and fault management techniques that best satisfy objectives defined in the Policy database.

Figure 2-5 shows the layout of the implementation database. The top level lists all task classes appearing in the policy database identified by task class name. Each task class name points to a list of alternate implementations. Associated with each implementation are its processor, communications, and data storage needs and the levels of performance and consistency it achieves.

Each task implementation instance is identified by a task name. Relevant information regarding task command line parameters is associated with each implementation. Also associated with each implementation are its processor, communications, and data storage needs, levels of performance and consistency it achieves, and a monitoring specification.

The processor, communications and data storage requirements and constraints associated with the task implementation are specified in the resource specification object. This specification is modeled to match the specification of the Resource database. In addition to specifying the hardware resources required by the implementation, the task implementation's resource specification identifies operating system, support software, and special communications interfaces which constrain the assignment of task imple-

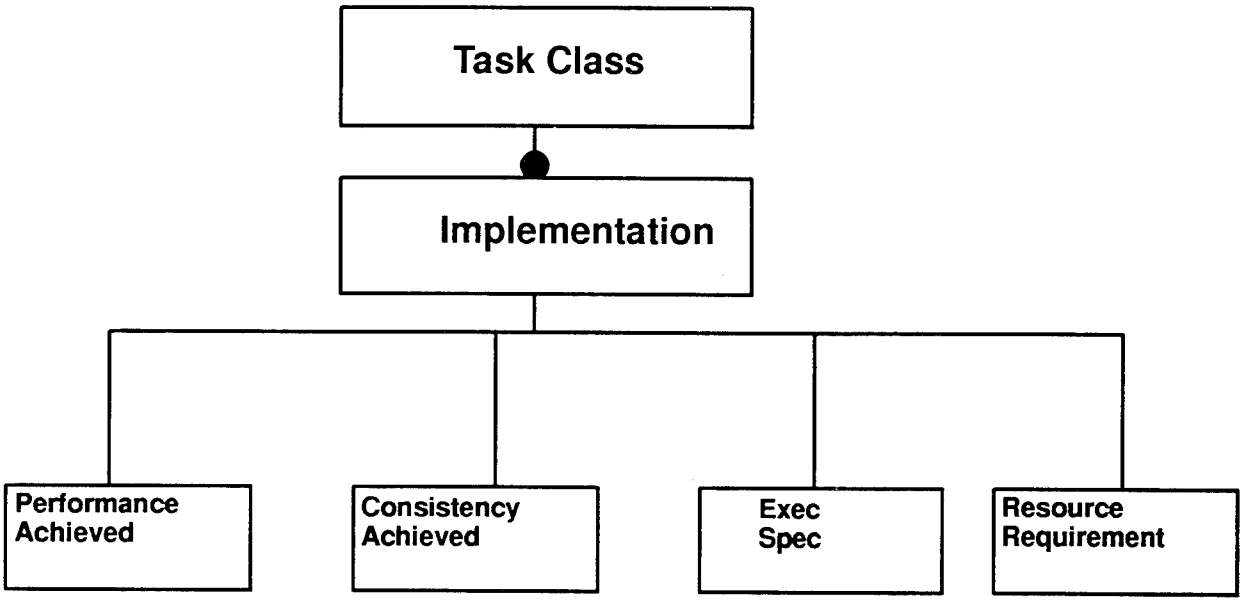


Figure 2–5 Diagram of the Implementation Database Structure

mentations to physical resources. The ARM uses the constraint information to form a first–cut assignment of task implementations to system resources.

Each task implementation is additionally described by an execution specification which lists pairs of executable code paths and the CPU architecture associated with the implementation. The Implementation database provides methods by which the ARM can determine if a task may be executed on a member of the CPU class. For example, a task which has an associated "Intel 80386" CPU is upwardly compatible with an "Intel 80486" CPU, but is not downwardly compatible with an "Intel 80286" CPU.

Finally, each task implementation has an associated monitor specification. The Internal State Monitor gathers generic system hardware and software performance, utilization, and fault history information as well as application specific information. The monitor specification provides a mechanism for the application architect to embed Meta sensors and actuators into a shell encapsulating the application. The application architect may choose to additionally insert Meta sensors and actuators directly into the application. The monitor specifications require that the data type of the monitored information be specified as well as an identifying name. Meta sensors can operate by periodic polling or on an event driven basis. The periodicity variable of the monitor specification al-

lows the application programmer to specify whether the sensor is periodic or event driven. Each Meta sensor is activated upon satisfaction of a guard condition. The guard condition and the action to be taken upon activation are specified respectively in the condition and action methods associated with the monitor specification object.

The Implementation database was designed to contain information regarding the impact of the various implementations of the task on the resources. This definition had to be expanded to include different types of resources. For example, how a tracking algorithm executed on a SPARC 2 versus a SPARC 10. The calculation of the impact of algorithms across platforms is still an active research question in universities and resource laboratories alike.

2.4.6.4 Resource Database

The Resource database provides the ARM with a model of the processing, communication, and software resources of the system. The Resource database is a static information repository of the potential resources available; the Resource database must be compared against the Internal State Monitor's Perceived Internal State data store to gain a true picture of the potential versus actual resource availability.

The resource database contains the following pieces of information for each resource: (1) CPU, (2) display, (3) storage, (4) communications, (5) software, and (6) operating system. Each of these classes is modeled with defining characteristics. For example, the CPU class is defined by an architecture , the potential number of millions of instructions per-second the CPU can execute, whether a floating point co-processor is present, whether on processor cache is available, and the clock rate of the processor. This set of information is certainly not the only information which might describe a CPU, but it is sufficient for the purposes of proof of AFRS concept demonstrations.

2.5 Enabling Technologies

The Internal State Monitor leverages MML's Distributed Resource Manager (DRM) IR&D work, the Isis and Meta distributed processing toolkits, the Cronus object-oriented environment and the ART-IM reasoning tool. The following sections provide introductory information about these enabling technologies.

2.5.1 Isis

Isis is a distributed processing toolkit that provides fault-tolerant communications capabilities for cooperating processes across heterogeneous platforms. Isis has been in development since 1984 at Cornell University, under DARPA sponsorship. It has been used successfully in many research projects, and recently has ventured into the commercial arena. Isis develops the notion of process groups, virtual synchrony, and broadcast capabilities in order to make it easier for the system builder to focus on the application, not the distributed communications substrate. The notion of process groups allow independent processes to belong to an Isis-maintained collection to which messages can be sent. By knowing the group identification, a process can send a message to the group, and Isis will multicast it to all members of the group. The notion of a process group is further enhanced by the virtual synchrony concept. Virtual synchrony is the guarantee that messages sent to a process group by different processes will be be seen by all group members in the same order. By providing this capability, the user need not concern himself with handling race conditions that can occur in a distributed environment. Exploiting Isis's ability to do detection and notification of group membership changes, it is possible to build replicated, highly fault tolerant applications.

In the FMT API which communicates with itself and with the ISM component heavy advantage of the atomic broadcast ordering provided by ISIS is used to guarantee that replicated tasks in the system see messages in the same order. This allows processing of messages in each instance of a task to proceed in the same order while not actually taking place at the same time in each instance which would be impossibly slow because of its distributed nature. Group membership change notification is one of two methods employed for failure detection within ISM.

2.5.2 Meta

Meta is a system built on top of Isis that allows application builders to monitor and control distributed applications easily. It uses Isis' distributed communications capabilities to provide the user with sensors and actuators. Sensors are objects that monitor values in the application. These values can be user specific, such as variables, or system-related, such as resource usage. When a value that is being sensed changes, Meta sends a message, via Isis, to the associated process group. Similarly, actuators trigger a function call in a process when a particular monitored value is reached. This is also done via Isis messages. Together, sensors and actuators can be used for distributed control of a user application or as a monitor of an application's internal state.

The user specifies how sensors and actuators perform using Meta's description language, Lomita. Lomita allows the user to specify sensors and actuators for each process, and a list of rules that can be used to determine when to fire actuators. By using Lomita, the Internal State Monitor can alert the AFM when the number of faults within a particular process reaches a threshold.

Currently the FMT API provides an interface for declaring monitorable pieces of information that hides the presence of META from programmers. Once declared in the code a description of the user monitorable data must be supplied in the Implementation Database. No visual method for monitoring user supplied data exists at this time.

2.5.3 Distributed Resource Manager

The Distributed Resource Manager (DRM) is an MML research program that exploits the benefits of Isis and Meta. It is an X-Windows-based monitoring tool that allows user-definable sensors and actuators to be specified and displayed in an easy-to-use graphical user interface. DRM uses the Jet Propulsion Laboratory's Widget Creation Library (WCL) to build control and monitoring widgets to be displayed on an X-Window screen. DRM combines the WCL with Meta to create a programmable user interface for a large distributed system. This technology was leveraged into the development of the Internal State Monitor by providing us with the ability to start and stop instances on remote hosts while also monitoring host level resource utilization.

2.5.4 ART-IM

ART-IM is a complete toolkit for the development of rule-based, or knowledge-based, systems. Available through Inference, Inc., it is the leading knowledge engineering tool in the nation. Version 2.5 was used on AFRS in developing the Allocation Function, Choose Next Pairing Function and the Planning Function in the Adaptive Resource Manager (ARM). ART-IM is used to execute the rules written for the functions listed above. For example in the Allocation function, rules were written to allocate tasks to resources, moving tasks from one resource to another and a method for degrading tasks.

2.5.5 Cronus

Cronus is a distributed object-oriented execution toolkit developed by BBN Systems and Technologies. This toolkit was used to build the system databases used in AFRS. The object-oriented paradigm was used to build the databases which are hierarchical in nature. Building the databases in this manner let the team update the structure of the databases more easily as the system was implemented.

3. Theater Missile Defense Application

3.1 Evaluation and Selection

Technology demonstration is an important part of the AFRS program. This section documents MML's effort to select a realistic BM/C3 application suitable for demonstrating the effectiveness of adaptive integrated fault management. The chosen application was picked from a set of ten candidates and was selected according to the criteria in the next section.

Considerations affecting the selection of an application for demonstrating adaptive fault resistance were collected from a number of sources. MML's proposal listed an initial set of selection criteria derived from the study objectives defined in Rome Laboratory's RFP. Both GE and subcontractor BCI added to this list during the beginning of the contract effort. Rome Laboratory's input was provided in the form of a Discussion Paper.

The expanded list of criteria was organized into six general categories shown below.

1. Relevance to Rome Lab's Survivable Distributed C2 Experiment
2. Minimization of cost and risk
3. Demonstration flexibility
4. Benefit from distributed operating system research
5. Benefit from adaptive fault tolerance research
6. Benefit from dynamic resource management

When a criterion pertained to multiple categories, it was assigned to a single one for convenience. The following subsections describe the considerations in each category.

In addition to the categorization, the selection criteria were prioritized to simplify the evaluation process. A few criteria were deemed essential, and failure to satisfy one of them justified elimination of an application. About half of the remaining criteria were ranked as high priority. To receive a strong evaluation, an application had to rate well against most, but not necessarily all, of them. The remaining lower priority criteria helped to make a final application selection. All of the criteria and their priorities are presented in the following subsections.

3.1.1 Relevance to Rome Lab's Survivable Distributed C2 Experiment

MML's Adaptive Fault Resistant System contract is an integral part of Rome Laboratory's Survivable Distributed C2 Experiment. The experiment seeks "to build upon the development of distributed operating systems, research in adaptive fault tolerance, and concepts in dynamic resource management to implement a survivable distributed C2 system for test and evaluation". Rome intends to expand the AFRS application through fiscal year 1995 to meet the experiment' s demonstration requirements.

Selection criteria related to the three cited technologies are contained in three corresponding categories below. These categories collect some required characteristics of the application domain. The focus is on the actual application rather than on a specific demonstration software implementation.

It was essential that the application is a distributed C3 System with real-time components. Although not mentioned in the description of the Survivable Experiment, network bandwidth was one of the resources to be managed dynamically for resilience to faults and graceful degradation.

A high priority criterion was that the application be in the domain of battle management. Some RFP references indicated a BM/C3 system, while others stated only C2. MM separated BM from C3 in evaluating several civilian applications that exhibit the important characteristics of military C3 but are not true battle management.

The criticality of functional survivability and continuity of operation was a lower priority criterion. The more important survivability is, the more justifiable is the investment in new fault management technology. Finally, it was not required, but certainly preferred, that the application be related to an Air Force rather than exclusively to an Army or Navy function.

3.1.2 Minimization of Cost and Risk

MML seeks to minimize the cost and risk associated with the application demonstration in order to devote as much effort as possible to the development of AFRS technology. As a starting point, it was essential to have a robust existing simulation of the application domain including its primary functions. A second essential criterion was the approval from the cognizant organization to use the existing simulation both at MML and at Rome Laboratory.

There were a number of high priority considerations. MML should already have a thorough understanding of and experience with the application domain. The availability of technical support was important for modifying the simulation. Even with a fairly complete system implementation, extensions were likely to be needed to simulate faults and errors and for integration with the adaptive fault manager's internal state monitor. Installation of the demonstration in Rome Laboratory's Distributed System Evaluation (DISE) Environment required the consideration of the existing system's programming languages, hardware, and operating system. With regard to Cronus Distributed Operating System support, the preferred languages were C, Ada, and C++. While not homogeneous, most DISE nodes are Suns running Unix.

Some lower priority cost and risk considerations were the complexity and quality of the existing application software and the suitability of its current HMI. Another factor was that running the simulation should not always require a large number of people and machines.

3.1.3 Demonstration Flexibility

This category combines a number of selection criteria intended to facilitate initial development at MML, future enhancements as part of the Survivable Experiment at Rome Laboratory, and open demonstrations at both sites. No considerations were prioritized as essential, but it was highly desirable that a realistic version of the application be implemented at the unclassified security level and that the source code be non-proprietary.

Some lower priority criteria were scalability and extensibility. The system should be scalable across different local configurations (such as numbers of processors and displays) both for fault tolerance and for demonstration convenience. It was desirable that a survivable version be easily scalable across a WAN or multi-cluster. Finally, the application should be extensible to accommodate additional fault management technology.

3.1.4 Benefit from Distributed Operating System Research

Rome Laboratory's Survivable Experiment intends to build upon the development of distributed operating systems. A single high priority criterion in this category was that the existing application software already be distributed. In the first category, the requirement was imposed that the application be an inherently distributed system. The

consideration here is that the existing simulation not merely represent a distributed system but itself have a distributed implementation.

Other criteria relate to benefit from the Cronus Distributed Operating System and the ISIS Toolkit. It was desirable that a more survivable or fault tolerant implementation of the application could be achieved by employing key infrastructure support provided by Cronus and by ISIS. These considerations were given a lower priority not because of their lack of importance but because many benefits of Cronus and ISIS will be realized directly in MML's adaptive fault manager design regardless of the application selection.

3.1.5 Benefit from Adaptive Fault Tolerance Research

This category contains characteristics that discriminate the potential of an application to take advantage of research in adaptive fault tolerance. There were a number of high priority selection criteria. For integration of fault management across the control dimension, the application scenario should transition through different modes or missions with different functionality, performance, and consistency objectives. The application system should be subject to realistic faults and errors, and it was a head start if the existing simulation already generates some of them. There should be some quantitative measures of system effectiveness to help characterize the costs and benefits brought to the system by the adaptive fault manager.

At a lower priority, it was desirable that the application fit the assumed fault model so that the granularity of faults and responses is of the order of a second and that the line replaceable modules are workstations. Beyond a collection of potential faults and errors, one or more catastrophic failure points that can be eliminated through fault management techniques would help emphasize the depth of the technology benefit. If a fault tolerance approach is already incorporated in the existing simulation, that was advantageous. Finally, it was helpful if the application had a diverse set of adaptivity alternatives such as parameters controlling processing, alternate function implementations, and alternate assignments of processes to hosts.

3.1.6 Benefit from Dynamic Resource Management

Selection criteria in this final category indicate the application's need for and potential to benefit from integrating fault management across the resource dimension. The first high priority consideration was that the processing, communication, and/or data storage resources were sufficiently limited and stressed that there is a real resource man-

agement problem. The application should have at least five to eight distinct modules so that the problem is reasonably complex. Besides faults and errors, the dynamics of the scenario should involve changes in configuration such as a variable network topology or the loss of processing or storage assets.

3.1.7 Rationale for Final Application Selection

Out of the ten evaluated candidates, the Theater Missile Defense Command Center (TMD or TMD-CC) met all the highest priority selection criteria and was evaluated as an excellent candidate for the AFRS demonstration for the following reasons:

a. TMD is already distributed and has many informative HMI screens on multiple workstations.

b. The Batman scenario driver for TMD is an interactive part of the system and collects meaningful quantitative measures of effectiveness.

c. The TMD demonstration already has three distinct modes of operation with different objectives. Moreover, TMD-CC already has alternate algorithms in place for weapon to target assignment.

d. Finally, TMD-CC's airborne defensive weapons make it more immediately relevant to Air Force operations.

3.2 TMD Description

The TMD Command Center (TMD-CC) provides BM/C3 support for a limited geographic area. The commander in the TMD-CC needs to be able to assess the threatening nature of countries in the theater and make command decisions during battle. During peacetime mode of operation, the commander's primary responsibility is to monitor and evaluate the military, political, economic, and social conditions of the countries in the theater and determine whether any country is gearing up for hostile activities. During crisis mode, when hostile activities are detected, the commander makes decisions regarding deployment of ground, air and sea based defensive weapons. During war time mode, the commander's responsibilities are to decide on Course of Action (COA), Tactics, Weapon /Threat assignments and Weapons Release Authorization (WRA). During the entire time of TMD-CC operation, information and decisions need to be relayed to and from the national Consolidated Command Center (CCC) and information gathering devices such as radar, satellites and mobile units; however, the TMD-CC needs to continue functioning even if communication links are disabled.

3.2.1 Scenario Evolution

The chosen TMD scenario is a theater in the Middle East with launches coming out of Iran towards Saudi Arabia. The demonstration runs at real-time and takes approximately 30 minutes to complete. The scenario begins with the TMD Command Center at a peace state. During peacetime mode of operation, constant country assessments are being made as to the threat of each country in the theater. The data for Iran is then updated incrementally to show an increasing aggression and to demonstrate the capabilities of the Country Assessment Screen. The updating continues up to the present moment, and the decision aids indicate that the aggressive country is at a state of war. The Battle Manager (BATMAN) simulator then begins to simulate the launch of incoming threats. As these threats are detected by the decision aids, they are also displayed on the commander's screen in tabular and graphical format. The decision aids then make recommendations to the commander to aid in the analysis of the situation and in the decision making process. A map display overlays relevant information in the theater such as known weapon sites, aircraft, and missile tracks. The commander must make decisions as to the placement of weapon aircraft and mobile sites, authorize weapon release, and inform areas of potential risk (passive defense). The BATMAN simulator calculates the launching of defensive weapons and deposits this information into the database, from which it is retrieved for display.

3.2.2 Existing Demonstration Content

The TMD Command Center HMI is made up of five basic screens: (1) Cinc Theater Commander Screen, (2) Event Indicator Screen, (3) Map Display Screen, (4) Threat Assessment Screen, and (5) Weapon-Target Assignment Screen. A printout of these screens is included in the Appendix of this document. Keep in mind that the screens are actually in color, so a true representation can not be made here.

The Cinc Theater Commander Screen displays assessments by the decision aids and performs all the command functions. The Event Indicator Screen shows the state of incoming threats. The Map Display Screen overlays weapon locations and missile tracks with the map of the theater. The Threat Assessment Screen assesses each country's threat. The Weapon-Target Assignment Screen shows weapon to target matching.

The modules which comprise the TMD-CC application include: (1) BATMAN, (2) Netrunner, (3) Adaptnet, and (4) Astrocalc. An SQL-like database is the central depository of all data and inter-process communication. BATMAN is the battle simulator which launches and propagates weapons and calculates hit or miss. It was developed by GE

in the 1980's as a space-based laser simulator. Over the course of several years, the model was modified to include kinetic energy weapons and ground-based interceptors. This is the module which drives the war mode of the scenario.

Netrunner is the control program which processes most of the decision-aid neural nets. There are over 35 neural networks which evaluate over a hundred data points during the simulation. Netrunner continually monitors the simulated world and propagates input data to the correct neural net. Some of the neural net responsibilities include threat assessments, decision aids, and weapon-threat assignments. The networks were developed by a tool developed by GE-CRD called GEN-R. This tool provides an Application Programmer's Interface to creating and running the neural networks. It also provides a textual front-end so that networks can be generated interactively and provides for ease of experimentation.

Adaptnet is the program which reconfigures and retrains the neural nets when conditions change. Traditional paradigms require re-training a neural net from scratch when a new decision, new input, or change in complexity arises. This can be cpu-intensive process and require time out of the bounds of a real-time system. Adaptnet solves this problem by using previously trained knowledge as a starting point and continues to train and grow the neural net as needed.

Astrocalc predicts impact points and calculates launch points based on information available about tracks. This is a highly cpu-intensive module and its resource requirements change as more threats are tracked.

3.2.3 Existing Demonstration Architecture

To achieve maximum performance, the applications and processes are distributed over the entire network of workstations. All communications between processes are performed through the database as shown in Figure 3-1. This data-driven architecture achieves process independence and requires no overhead of registering with a master control program. An application can be added, deleted, or changed with no impact to the rest of the command center. This provides for future upgradability as new technology is incorporated or as the source of the information changes (i.e., real-time radar information could be fed in and all applications would have the new data). To reduce the possibilities of a database bottle-neck, the database is placed into a ramdisk so that it will operate at cpu-speeds. Each application in the command center is independent of the others and is capable of being shut down and restarted. The baseline TMD-CC does not employ any fault management or fault detection techniques.

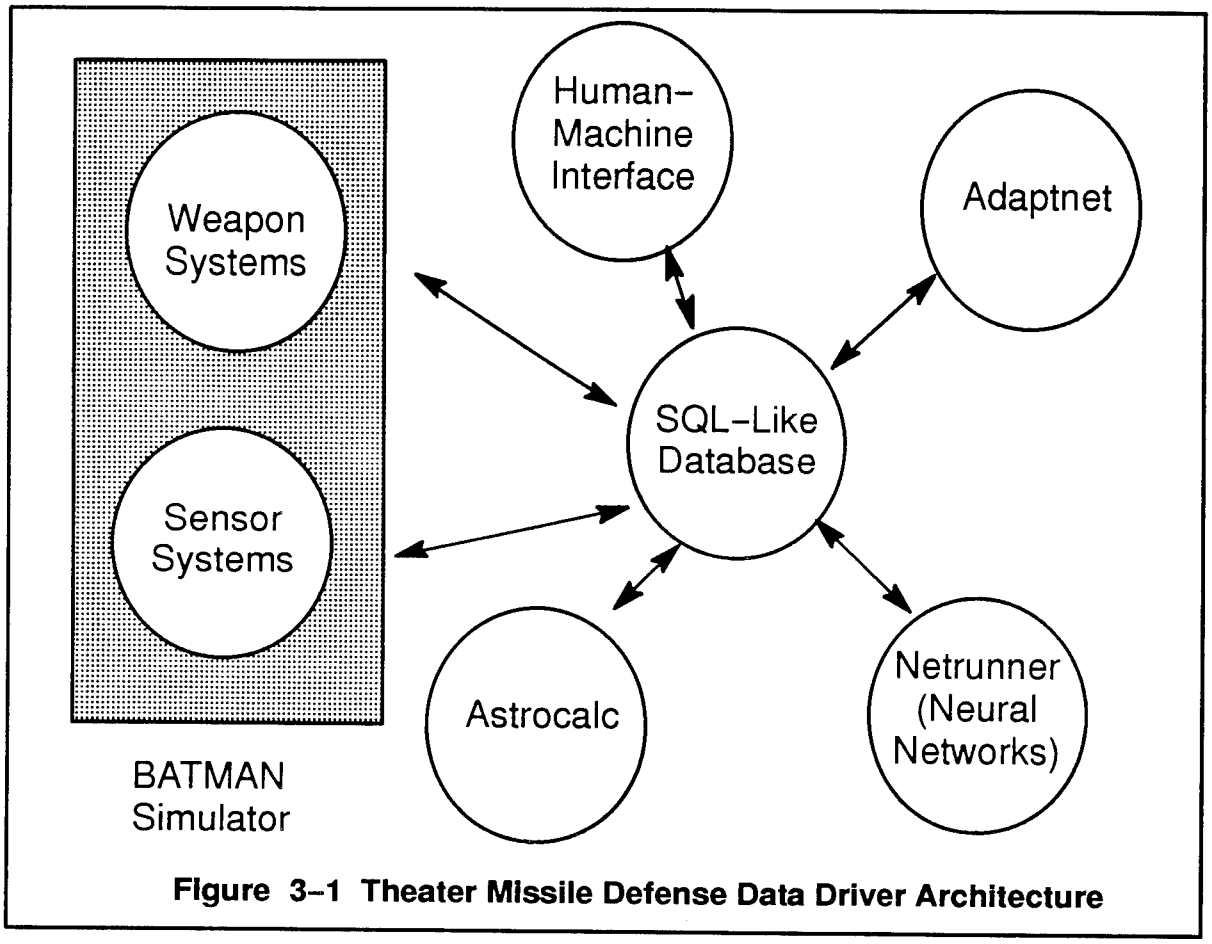


Figure 3–1 Theater Missile Defense Data Driver Architecture

The scenario is driven by the BATMAN simulator. BATMAN calculates trajectories and interceptions based on probability of kill of individual weapons. It simulates sensor detection by placing the current location of all incoming threats and outgoing weapons into the database. All other applications read the database and process this information. Since BATMAN's simulated sensors detect incoming threats at cloud break, Astrocalc is needed to calculate the launch points based on the current location and velocity. Astrocalc also predicts impact points and deposits them into the database. These predictions, along with the current location of threats, are used by the weapon–target assignment neural nets to calculate the best allocation of weapons.

3.2.4 Hardware and Software Requirements

The hardware and software requirements for run-time operation are outlined in Figure 3-2. All MML developed command center applications have source code included. The only COTS package which includes non-proprietary source code is the Transportable Application Environment (TAE). This is a tool to generate graphical interfaces in the X-Window environment using a point-and-click interface and is available from NASA.

	Item	Specification
Hardware	Workstations Ram Virtual Memory Disk Space Network	4 Sun Sparcstation 1+'s 96 Mbytes (32+24+24+16) 192 Mbytes (64+4848+32) 200 Mbytes Ethernet Local Area Network
COTS Software	Operating System Libraries HMI System Decision Support Sys.	SunOS 4.1.3 (UNIX) Motif1.1, X11R4 (Openwindows) TAE GEN-R

Figure 3-2 TMD-CC Hardware and Software Requirements

4. Testbed Description

4.1 Hardware

The AFRS testbed has been installed at the DISE facility at Rome Laboratory and has been named the Survivable Adaptive Fault-Tolerance Experiment Network (SAFTE-Net). The hardware was setup by Rome Lab system administrators per the requirements by MML's AFRS team to run the AFRS system and the application. SafteNet consists of four Sun Workstations networked together by an ethernet lan segment. The following table lists the hardware configuration of the net.

Machine Name	Machine Type	Monitor	Disk	Memory (Ram)
pizza1	SparcStation 1+	17" Color	525 Mbytes	32 Mbytes
pizza2	SparcStation 1+	17" Color	525 Mbytes	16 Mbytes
pizza3	SparcStation 1+	17" Color	525 Mbytes	16 Mbytes
pizza4	SparcStation 1+	17" Color	525 Mbytes	24 Mbytes

4.2 Software

The software base for the testbed is a combination of COTS packages that are readily available and all efforts have been made to use software that adheres to industry standards and open systems. The following list includes the software required for the AFRS system and the TMD application to run correctly. All packages are available on each machine except for ART-IM which only required one run-time license.

1. ART-IM, v2.5, rev. 2
2. SunOS v4.1.3
3. Openwindows v3
4. X11R4 (included in Openwindows)
5. Motif v1.1
6. Isis/Meta v3.0.8
7. TAE
8. Cronus v2.0

9. C compiler (included with SunOS 4.1.3)
10. Fortran compiler (SC1.0)

5. Lessons Learned

Lessons learned from each section of the program are discussed in the sections following. A general accomplishment of the entire system, however, is the ability of switching fault tolerant techniques. Before the AFRS system, systems that implemented fault tolerant techniques fell back to restart methods when failures in the system occurred. The AFRS system however, allows the system to adapt between fault management techniques. This is not to say that every technique may be adapted to any other technique but options other than restart are available. For example, a voter may be adapted to a primary backup technique.

5.1 Internal State Monitor (ISM)

The most difficult problem discovered in the implementation of the ISM was the transfer of application state information to new instances of a task. The transition of state is an extremely difficult problem which requires the cooperation of the programmer since no automatic way could be found. Not only does the instance need to communicate with the ISM component the times when a state transfer may occur but it must also be capable of transferring the information. ISM and the FMT API require an interface with the programmer through which this may be accomplished.

The user must be made to understand what information is essential to transfer state. The user needs to record all data and system information that is needed to bring a new process up to date with what has occurred thus far in the process. The state must then be packaged up for transition.

The state internal to the AFRS system was very difficult to encapsulate also partly because of its size. If a voting FMT is in use each message which hasn't been voted upon from each instance must be sent. If primary/backup is being used messages which the backup hasn't seen the primary send but has been locally processed must be sent as well as messages which the primary has generated but the backup hasn't processed must be sent. This transfer of information is invisible to the user as it should be. The hardest problem here however was the numerous special cases or race conditions that exist in a hostile environment while attempting to switch FMT modes or start new instances of tasks. This was expected to be a hard problem and fulfilled every expectation of difficulty.

The user must also establish points in the process in which the state may be transferred effectively. For example, a calculation must be finished or in a stable state, or the results may be lost during state transfer.

The communication with the user of the above information lead to the development of the FMT API, also known as the client library. Initially, development of the idea of a truly transparent library was considered for the FMT API. Even with cooperation of the operating system this was deemed impossible because several important pieces of information were unavailable to the system. Examples include the voter to use as well as which pieces of information need to be transferred or not transferred. See section 2.4.5 for more information.

The FMT API offers the programmer a common interface for all fault management techniques and for the adaptation between them. This API offers all the services needed for implementation of the fault management techniques and a convenient, easy to use, message passing and remote procedure call interface. This interface is identical regardless of the FMT that is currently applied to the process.

5.2 Adaptive Resource Manager

The Adaptive Resource Manager (ARM) is a highly data driven, data intensive system. The basic requirements of this portion of the program require a highly organized, efficient and effective method for manipulating the data involved. The basic data types dealt with include: resource descriptions, implementation descriptions, fault management technique descriptions, descriptions of the current system state and desired system states. The ARM is required to place a task (a combination of an implementation with a fault management technique) onto resources. A task may be composed of many processes in the case of a multiple version or replication fault management technique. Each task, resource and individual process must be uniquely identified so that the ISM and the ARM may pass messages back and forth communicating the states of the resources, tasks, and changes that occur. This insures that the instructions given by the ARM are executed on the correct process.

In this implementation of the ARM, the division of work was divided into three sections: State Space Searcher, Resource Allocation, and System Adaptation Planning. Each is an independently functioning portion. However, over the course of the development, it

was found that the State Space Searcher and the Resource Allocation modules worked better in a more tightly coupled environment. In fact, to achieve a more optimized search, ideally the searcher and the allocator should be even more tightly coupled than they are in the current system. By coupling the two functions, it allowed for feedback regarding the system state and what implementation may have a "better" chance of achieving allocation. By introducing a limited amount of feedback, the process can be improved greatly by communicating to the searcher the resources that are currently under the greatest usage. This allows the searcher to choose those implementation/FMT pairings which least impact these resources. The effects of implementing the feedback between the searcher and the allocation process has been studied under the AFRS contract, however was not implemented. In future extensions of the system, the change will be added.

A difficult problem, however, is the calculation of the impact of the various implementations of the tasks on different resources. ARM needs to make accurate prediction of how implementation use a resource. This requires that an empirical study be made on each of the implementations that are going to be used in the application as well as a study of the fault management techniques. These values are then inserted into the implementation databases and the fault management techniques database (i.e. the amount of memory used by a processes or the amount of overhead added by a fault management technique). The ARM bases the models of the resources and the tasks on these number, thus if the values in the databases are wrong, then the recommendations of the ARM will be in error. This may cause the ARM to recommend a course of action that can not be implemented. The University of Maryland studied the possibility of a formula that might be used to predict the effects of an implementation across platforms given statistics on another platform. However, this questions still requires a large amount of research before it could be implemented in a reliable manner.

A related issue is the impact of the fault tolerant techniques on the execution of the various implementations. This requires a large empirical study of the impact of the various techniques on the implementations. Such studies would improve the allocation process.

Another method for improving the ARM would be to define heuristics for the implementation/fault management technique pairings. This is a very difficult problem in that the definition of heuristics require a formalization of the requirements of the implementation and fault management techniques. Heuristics of this nature are most often

expressed in terms of a mathematical formula. As discussed above, the formalization of the impact of implementations and fault management techniques on resources needs much more research before it could be realized However, if such a formalization was available, it would allow us to "weed out" or change the method in which we select the faults management techniques and implementations that would make the ARM more efficient.

A critical assumption made at the beginning of AFRS was that the application would be executed on a fully connected network and NFS mounted system. This allows the assumption that each task will be able to communicate without having to consider routing or if the nodes are connected. This allows the assumption that if a file is shared among implementations that it will be accessible to the implementations no matter which resource/node the implementation will be executing on. Without this assumption, the complexity of the ARM more than doubles because of the calculations that would be required for analysis of the implementation requirements of network topology, connectivity. The information that would be required for this analysis would be: the network topology of the resources, an internal representation of the topology, the connectivity of the application in terms of processes, and connectivity requirements the FMT would have on the processes (for example in the cases of multiple version or replicated processes). The information would be needed during the allocation process so that the ARM could guarantee the connectivity required by the tasks. This type of analysis has been done on other programs, however, it is beyond the scope of the AFRS contract. The ARM could be expanded to include this type of analysis in the future.

The System Adaptation Planner function is relatively straight forward with the exception of the case in which the system resources are near capacity. In this case, the planner may enter a situation in which there is no solution. This is particularly a problem when the processes require that the previous copy of the process must be maintained for state transfer or smooth transition of messages. In situations in which all processes must be transferred in this manner, a single process may need to be stopped in order for the planner to complete the transition.

As stated previously, coordination is required between the ISM and the ARM. This is especially evident in the transition between objective functions. This transition must be orderly and coordinated. To execute the transition between objective functions, the ARM must analyze what is required of the new objective function, compare this to the old objective function and make modifications to the current state of the system as

needed to satisfy the new objective function. These changes may include changes in priority of tasks, switching fault management techniques, bringing new tasks up or stopping currently executing tasks. The ARM must communicate these changes to the ISM. This however, requires that the ARM and the ISM correlate the identifiers of the process currently running and what they will be identified as in the future. As the identification may have changed during the transition. It is particularly critical for the ARM component as the identifiers are critical in the system in that each task is uniquely identified. These identifiers are required for coordination in switching objective functions, changing fault management techniques or switching implementations.

5.3 System Databases

The main theme with the implementation of the databases was the requirement for encapsulating changing requirements in a stable database.

The Policy database needed to be defined in a manner which allowed the user to represent requirements based upon a varying environment. The database had to allow the user to represent the desired actions for the different states of the world and the compromises that could be undertaken given a degraded environment. The concepts of performance, consistency and fault management requirements were difficult to define. However, definitions assumed by the AFRS system were workable. For a detailed description of the policy database, refer to section 6.4 of the Software Design Document (CDRL A006).

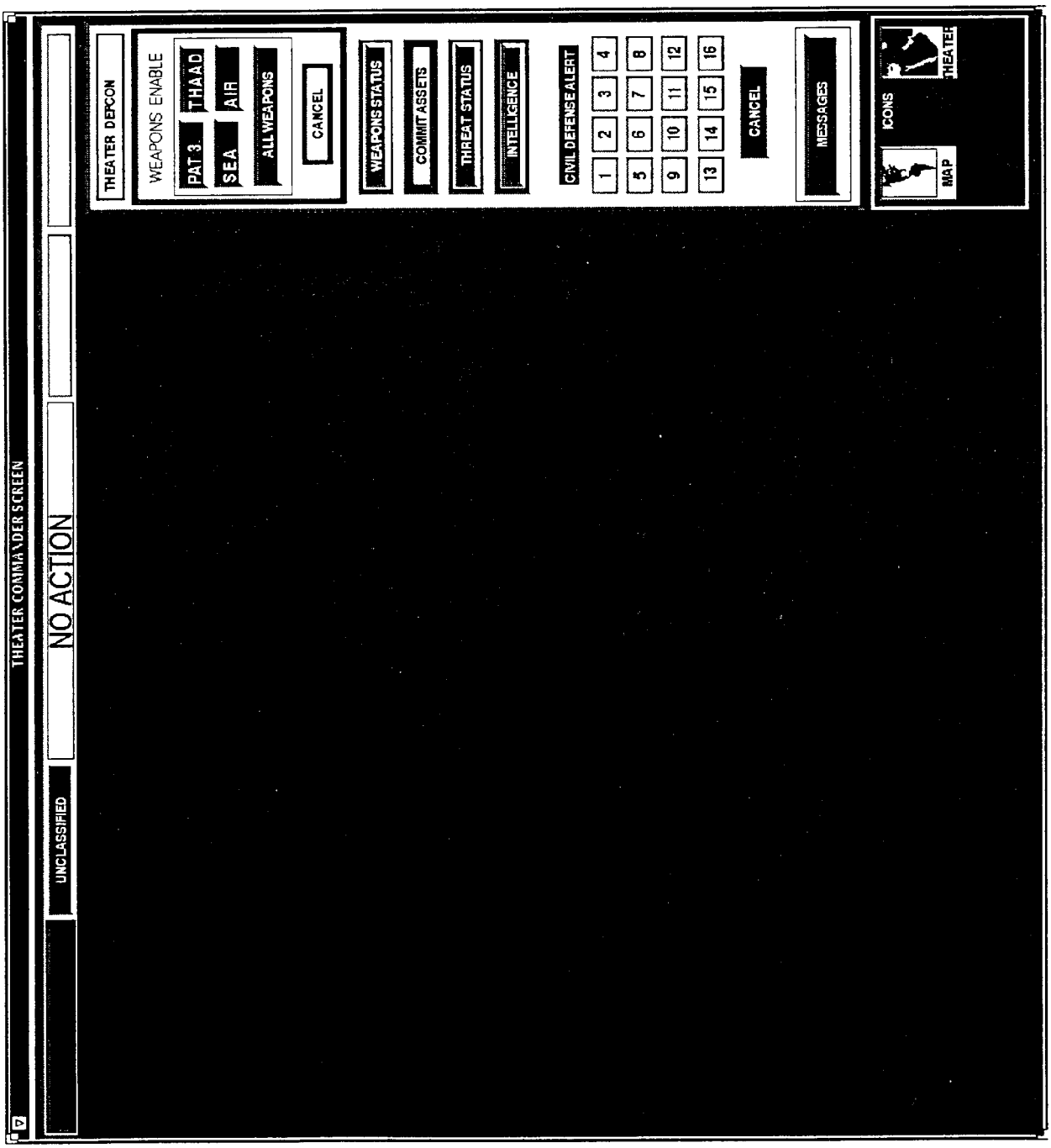
The Fault Management Techniques database has to also record variable data in stable fields. In this case the impact of the various techniques on processes. This information was stabilized by using the maximum impact of the FMT on the processor, memory and communication on the process.

A mapping of fault management requirements in the policy database to specific implementations was studied, but was not implemented in this version of the FMT database.

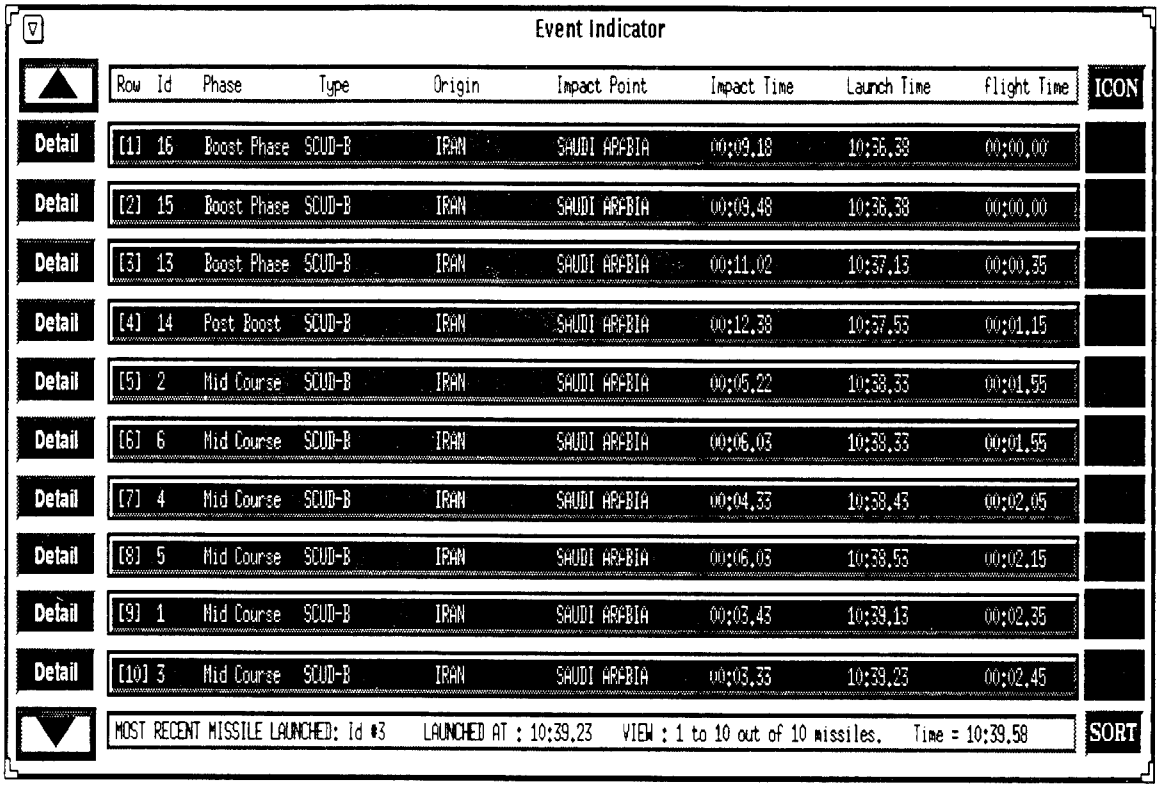
The system databases were designed to take information from a historical system. However, this was not implemented in the current AFRS system.

6. Appendix

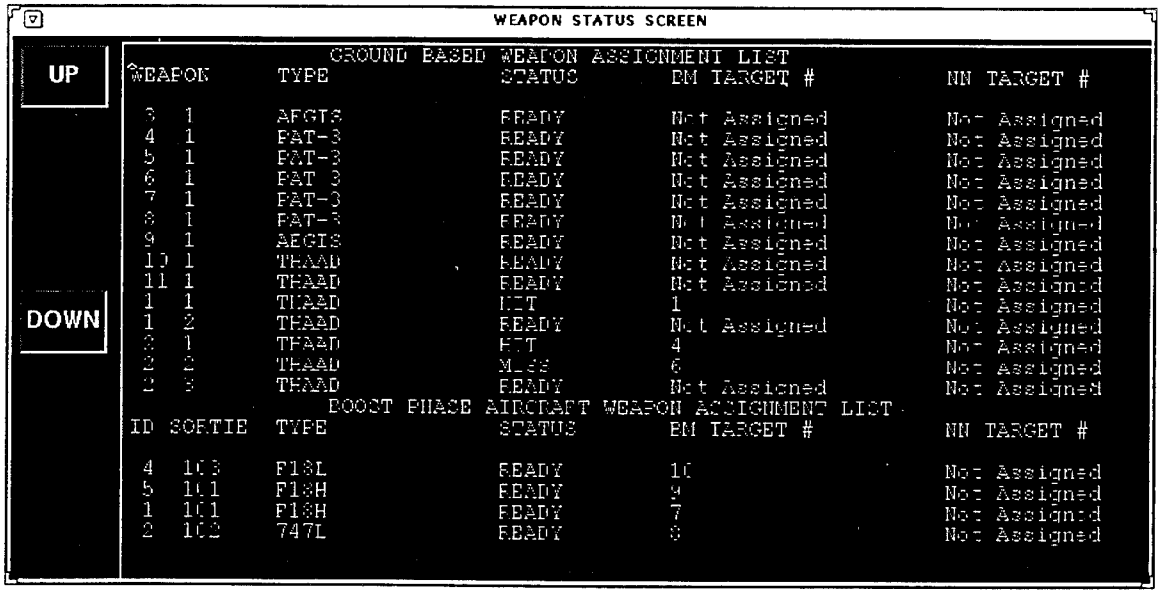
6.1 Cinc_Theater Screen



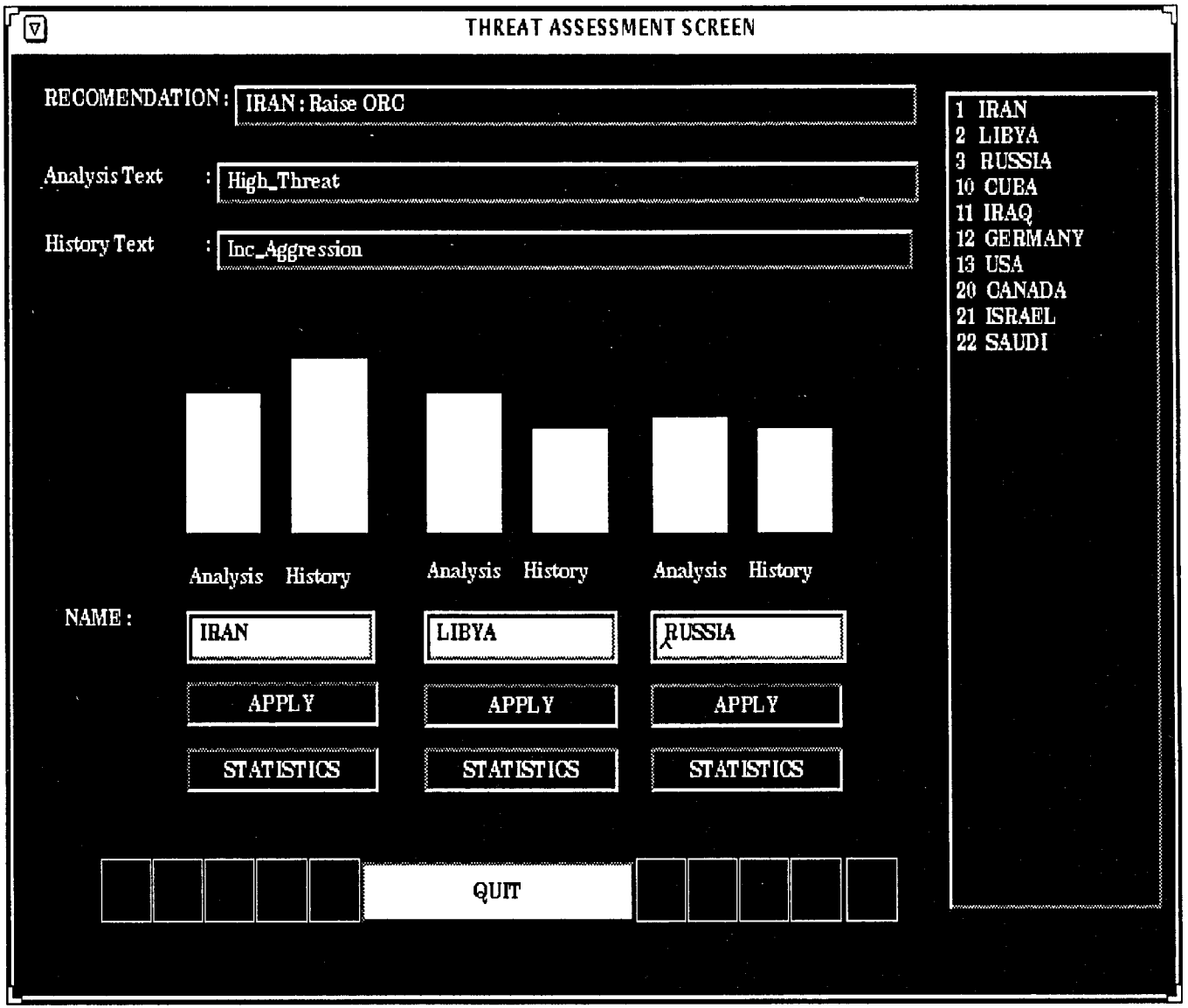
6.2 Event Indicator Screen



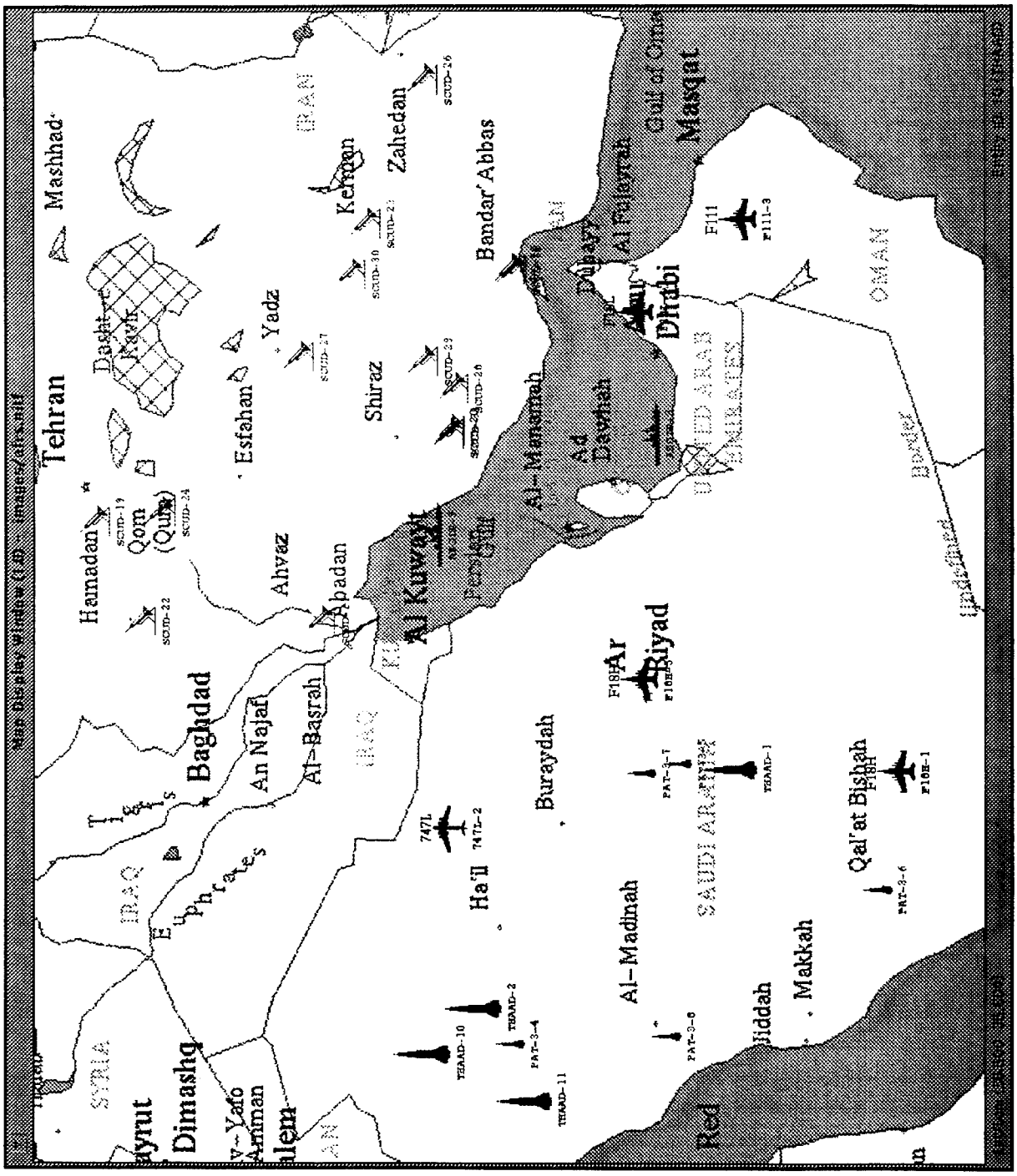
6.3 Weapon–Target Assignment Screen



6.4 Threat Assessment Screen



6.5 Map Screen



*U.S. GOVERNMENT PRINTING OFFICE: 1995-610-126-50137

MISSION

OF

ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.